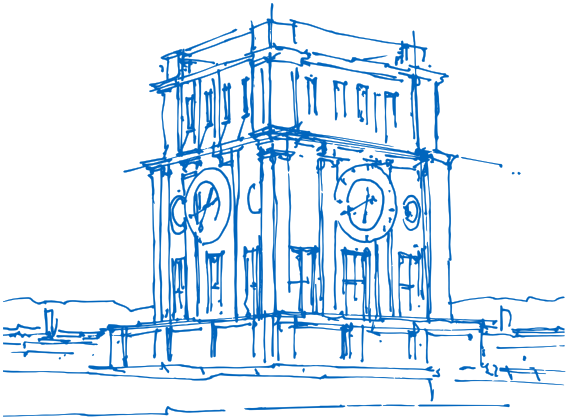
School of Computation, Information and Technology
Technische Universität München

Übung 05: Caches

Einführung in die Rechnerarchitektur



Michael Morandell

School of Computation, Information and Technology
Technische Universität München

18. – 24. November 2024

Mitschriften & Infos

Montags:
https://zulip.in.tum.de/#narrow/stream/2668-ERA-Tutorium—Mo-1000-4

Donnerstags:
https://zulip.in.tum.de/#narrow/stream/2657-ERA-Tutorium—Do-1200-2

Website: https://home.in.tum.de/ momi/era/

Keine Garantie für die Richtigkeit der Tutorfolien.

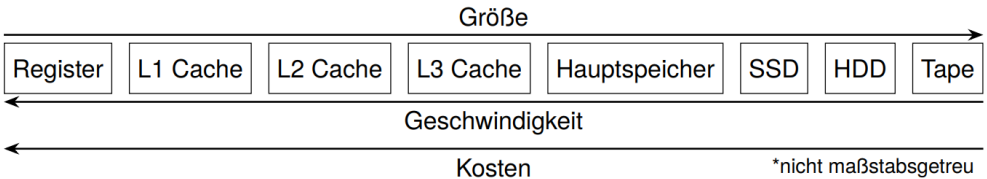
Bei Unklarheiten/Unstimmigkeiten haben VL/ZÜ-Folien recht!

Inhaltsübersicht

- Wiederholung
- Tutorblatt
  - Speicherzugriffszeit
  - Vergleich der Cachearten
  - Cachestruktur und Cachemisses
  - Ersetzungsstategien

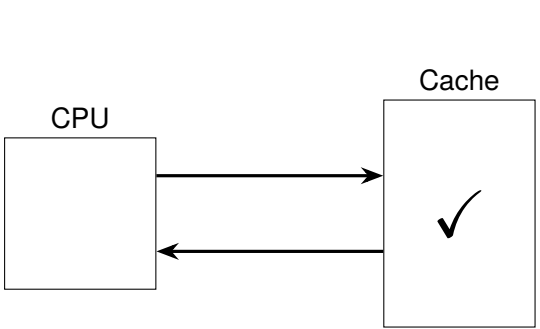
Motivation Caches

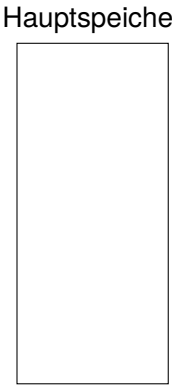
- Zugriffe auf Hauptspeicher ($\equiv$ RAM) sind **extrem** langsam. Lösung: Caches
- „Zwischenstation“ zwischen Registern (sehr schnell, sehr klein) und Hauptspeicher (sehr langsam, sehr groß)
- Idee: Häufig genutzte Daten im Cache zwischenspeichern, der Rest wird bei Bedarf aus dem Hauptspeicher geholt
- heutzutage meist L1/L2/L3-Caches: Caches aufsteigender Größe, aber absteigender Zugriffszeit



Cache-Terminologie

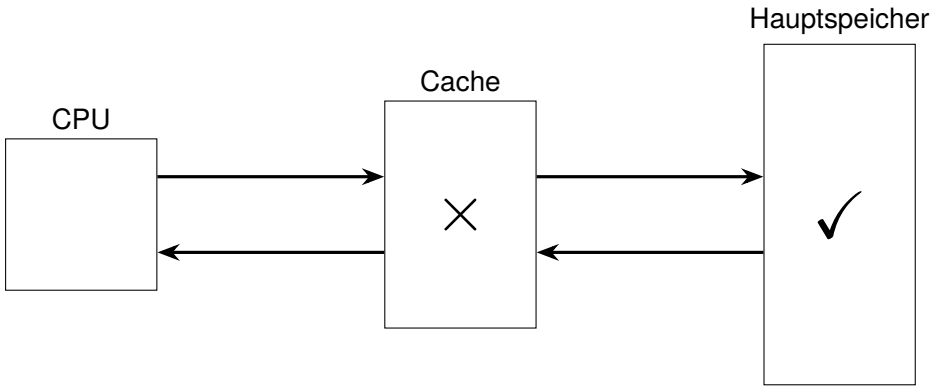
Hit Latency: Antwortzeit, wenn Wert im Cache





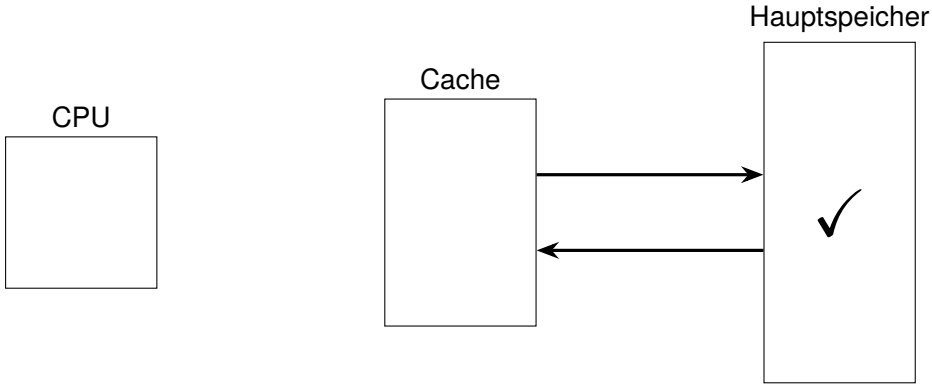
Cache-Terminologie

Miss Latency: Antwortzeit, wenn Wert nicht im Cache



Cache-Terminologie

Miss Penalty: Miss Latency − Hit Latency



Terminologie

- **Hit**: Datum liegt im Cache
- **Miss**: Datum nicht im Cache, muss erst aus Hauptspeicher geholt werden
- Ziel: möglichst hohe **Hitrate** (Hits/Anfragen) $\rightarrow$ häufig genutzte Daten im Cache
- **zeitliche Lokalität**: Zugriff auf x $\rightarrow$ wschl. Zugriff auf x in Zukunft
- **räumliche Lokalität**: Zugriff auf x $\rightarrow$ Zugriff auf Daten in der Nähe (oft durch Cacheline abgedeckt)

Messung der Cache-Güte

$$\text{CPU Time} = \text{IC} \cdot \left(\frac{\text{CPI}}{f} + \frac{\text{Memory Accesses}}{\text{Instruction}} \cdot \text{Average Memory Access Time} \right)$$

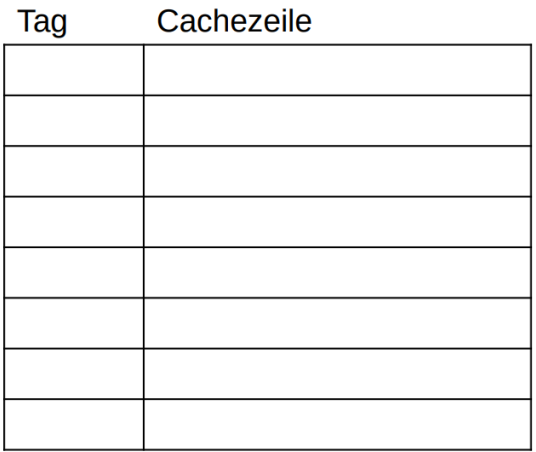
$$\text{Average Memory Access Time} = \text{Hit Rate} \cdot \text{Hit Latency} + \text{Miss Rate} \cdot \text{Miss Latency}$$

Frequenz (f):	Frequenz des Prozessors[1]
Instruction Count (IC):	Anzahl an auszuführenden Instruktionen
Cycles per Instruction (CPI):	(Durchschnitts-) Anzahl an Zyklen pro Instruktion
Memory Access Rate:	(Durchschnitts-) Anzahl an Speicherzugriffen pro Instruktion

[1] für uns in der Einheit $\frac{\text{cycles}}{\text{s}}$

Aufbau eines Caches

- Menge an Speicherzeilen einer festen Länge (Cachezeilenlänge)
- Jede Speicherzeile ist mit dem Tag, einem Teil der Adresse identifiziert
- In einer Cachezeile steht immer ein Block an Daten, die an fortlaufenden Adressen im Speicher stehen



Adressierung

- 4-fach-assoziativer Cache
- Cachegröße: 128 Byte
- Cachezeile: 4 Byte

$$\texttt{0x02C8} \quad \Rightarrow \quad 00000010110\ 010\ 00$$

Adressierung

- 4-fach-assoziativer Cache
- Cachegröße: 128 Byte
- Cachezeile: 4 Byte

$$\texttt{0x02C8} \quad \Rightarrow \quad 00000010110\ 010\ \underbrace{00}_{\text{Offset}}$$

Offset Bits – bezeichnen das Byte der Cachezeile, auf das zugegriffen werden soll.
Die Cachezeilengröße ist in unserem Beispiel 4 Bytes, deswegen genügen $\log_2 4 = 2$ Bits.

Adressierung

- 4-fach-assoziativer Cache
- Cachegröße: 128 Byte
- Cachezeile: 4 Byte

$$\texttt{0x02C8} \quad \Rightarrow \quad 00000010110 \underbrace{010}_{\text{Index}} 00$$

Index Bits – bestimmen das Cache-Set, in dem die gewählte Adresse sein kann.
In unserem Beispiel ist der Cache 128 Byte groß, hat eine Cachezeilen-Größe von 4 Byte und ist 4-fach-assoziativ. Also es gibt $\frac{\frac{128}{4}}{4} = 8$ Sets. Daher benötigt man hier $\lceil \log_2 8 \rceil = 3$ Bits. Die Daten werden also in Set #2 landen.

Adressierung

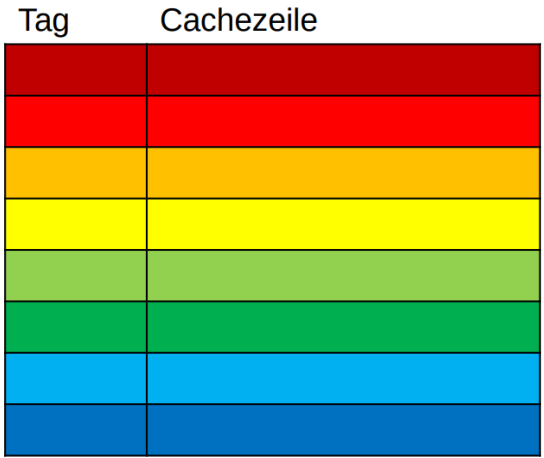
- 4-fach-assoziativer Cache
- Cachegröße: 128 Byte
- Cachezeile: 4 Byte

$$\texttt{0x02C8} \Rightarrow \underbrace{00000010110}_{\text{Tag}}\ 010\ 00$$

Tag Bits – restliche Bits. Identifizieren die in einer Cachezeile gespeicherten Daten eindeutig.

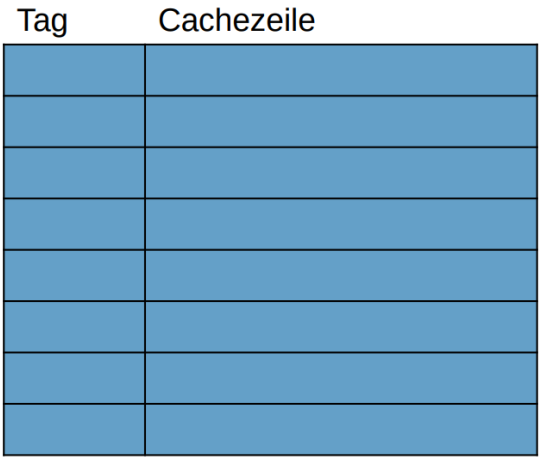
Cachearten: Direct Mapped Cache

- Ein Datum kann auf eine Cachezeile gemapped werden
- Wenn schon etwas drin steht, wird es verdrängt → möglicherweise sehr viele Verdrängungen
- Sehr schnelle Zugriffe → nur ein Tag vergleich notwendig



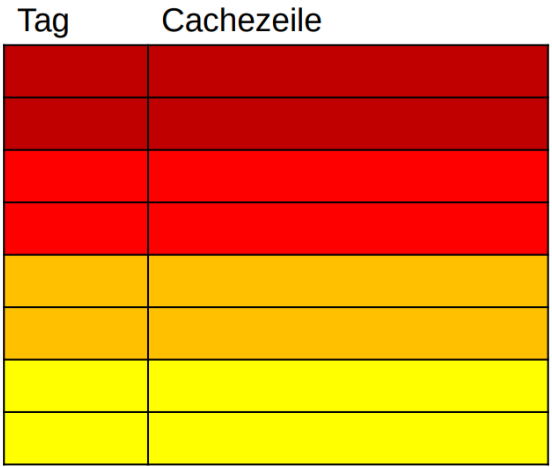
Cachearten: Vollassoziativer Cache

- Ein Datum hat keine feste Position im Cache, also keine Index Bits $\rightarrow$ wenn der Cache leer ist, laden wir einfach nacheinander die Daten in den Cache
- In jeder der Zeilen könnte das Datum stehen, auf das wir zugreifen wollen
  - Wir müssen bei jedem Zugriff schauen ob der Tag, auf den wir Zugreifen wollen, schon im Cache steht
  - sehr viele Vergleiche



Cachearten: Mengenassoziativer Cache

- Ein Datum kann auf eine Teilmenge der Cachezeilen gemapped werden (Index)
- Bei Zugriff müssen die Tags innerhalb der Menge angeschaut werden $\rightarrow$ relativ effizienter Zugriff
- Verdrängung eines Datums nur, wenn die Menge voll ist $\rightarrow$ weniger Verdrängungen, und wenn verdrängt wird, dann ein Datum, dass länger nicht gebraucht wurde
- Kombination aus Direct-Mapped und Vollassoziativ



Übersicht: Berechnung der Bits für Tag, Index, Offset

Vollassoziativ:
Anzahl der Cachezeilen: Cachesize / Zeilenlänge
Index = 0
Offset = $\log_2(Cachezeilenlänge)$
Tag = Adresse – Offset

Direct Mapped:
Anzahl der Cachezeilen: Cachesize / Zeilenlänge
Index = $\log_2(Anzahl\ der\ Cachezeilen)$
Offset = $\log_2(Cachezeilenlänge)$
Tag = Adresse - Index – Offset

Mengenassoziativ:
Anzahl der Cachezeilen: Cachesize / Zeilenlänge
Index = $\log_2(Anzahl\ der\ Cachesets)$
Offset = $\log_2(Cachezeilenlänge)$
Tag = Adresse - Index – Offset

Berechnungsbeispiel: Cache-Bits

Auf einer 32-Bit Architektur sei ein Cache von 32 Byte, mit 4 Byte Cachezeilenlänge gegeben.
Wieviele Cachezeilen gibt es ?
32/4 = 8 Cachezeilen

Was sind Tag, Index und Offset, wenn:

a) Der Cache vollassoziativ ist?
   Index = 0, Offset = log(4) = 2, Tag = 32 – 2 = 30
b) Der Cache 4-fach assoziativ ist?
   Index = log(2) = 1, Offset = log(4) = 2, Tag = 32 - 1 – 2 = 29
c) Der Cache direct-mapped ist?
   Index = log(8) = 3, Offset = log(4) = 2, Tag = 32 – 2 - 3 = 27

Berechnungsbeispiel: Cache-Mapping

Gegeben sei ein Cache von 32 Byte, mit 4 Byte Cachezeilenlänge.

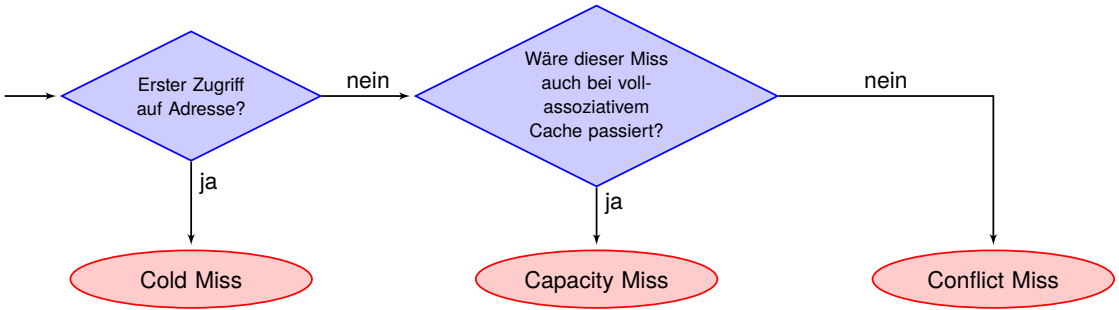
Auf wieviele Cachezeilen kann das Datum 0x7 gemapped werden, wenn

a) Der Cache vollassoziativ ist?
   Auf alle, also 8
b) Der Cache 4-fach assoziativ ist?
   Auf eine Cachemenge, also 4
c) Der Cache direct-mapped ist?
   Auf eine Zeile, also 1

Klassifikation von Misses

- Cold Miss:
  - Die Daten an dieser Adresse werden zum ersten Mal in den Cache geladen.
- Conflict Miss:
  - Die Daten an dieser Adresse waren bereits im Cache, wurden aber verdrängt, obwohl noch Platz im Cache war.
- Capacity Miss:
  - Die Daten an dieser Adresse waren bereits im Cache, wurden aber verdrängt. Dies wäre selbst bei einem vollassoziativen Cache geschehen.

Graph: Klassifikation von Misses



Ersetzungsstrategien

- LRU (Least Recently Used)
  - Löscht die Zeile, auf die am längsten nicht zugegriffen wurde
  - Gute Leistung bei vielen sequenziellen Zugriffen
  - Schlecht bei zufälligen oder verstreuten Zugriffen
- LFU (Least Frequently Used)
  - Löscht die Zeile, auf die am seltensten zugegriffen wurde
  - Gute Leistung bei repetitiven Zugriffsmustern
  - Schlecht bei ungleichmäßigen Zugriffsmustern
  - Erhöhter Overhead durch Zählerverwaltung
- FIFO (First in, first out)
  - Löscht die zuerst hinzugefügte Zeile

Fragen?

Bis zum nächsten Mal ;)

Folien inspiriert von Niklas Ladurner & ZÜ